Cost Reduction Through the use of Web Based Applications in the Mission Operations Center

Michael Packard[i], *Dennis Whichard*[ii], *P. J. Clark*[iii]

i The Johns Hopkins University / Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
Michael.Packard@jhuapl.edu

ii Interface & Control Systems, INC
8945 Guilford Road, Suite 120
Columbia, MD 20146
dennis@interfacecontrol.com

iii Patrick Clark
Dewitt & Associates
JHU/APL MS M6−110
11000 John Hopkins Road
Laurel, MD 20723
240−228−7666
patrick.clark@jhuapl.edu

Abstract

Even though many organizations utilize Commercial off the Shelf (COTS) products as the core of their mission operations and control centers, the user is almost invariably required to produce ancillary and interface software in support of the mission operations team. Although many of these custom products are reusable from mission to mission, each mission is usually unique in one or more aspects that result in some software design and deployment effort.

We are implementing the use of a web based design paradigm, which utilizes Open Source Software/Free Software (OSS/FS) as the basis for these products in the TIMED Mission Operations Center. We have found that this paradigm can substantially reduce the cost of the development, use and maintenance of this type of software and at the same time greatly facilitate its reuse in multiple missions.

We will describe the design of 3 web based products for use in the mission control center and specifically in the areas of conducting mission operations, data processing and archiving, and spacecraft contact and ground asset scheduling.

There exists a plethora of OSS/FS operating systems, relational databases, server applications and scripting languages that can be used as the basis for the proposed software applications. It should also be noted that the hardware cost to implement the designs we are proposing is quite modest. Furthermore, the use of scripting languages such as PHP and Perl, with their rich libraries for text parsing, communications and database manipulation, considerably speed development time and can result in applications that can be easily maintained.

The use of a web based paradigm replaces the development of GUI based applications using X or Microsoft MFC or other GUI development systems since all user interfaces are implemented via web browsers. This in itself can considerably reduce development costs since all GUI's are implemented using HTML and possibly Javascript. Also, in the case of the mission operations application we intend to describe, the cost of COTS licenses could potentially be reduced since fewer licenses may be required.

Since this proposal calls for the use of web based applications, it can be seen that access to the applications by the user would not necessarily have to be conducted only in a mission operations center. Through the use of Virtual Private Networks, Secure Sockets Layer, or other secure protocols, access to the applications could be conducted from a user's office or even off campus allowing the mission operations team to be in geographically diverse locations. This could also result in considerable cost savings. Also, since the interface to the applications is via a web browser, an interface quite familiar to most potential users, training costs and even documentation costs could be considerably reduced.

The 3 systems we intend to describe are in use or being developed for the TIMED Mission:

1) A web based data processing and archiving system.

2) A web based spacecraft contact and asset scheduling system.

3) A web based interface to COTS mission operations products.

1. Introduction

Web Based Applications (WBA) are software applications that interact with the user using traditional web pages or forms on a web browser. Using WBAs can reduce the cost of developing software used in mission control and operations centers. This interface is quite familiar to people in spacecraft operations as well as those developing the applications. They can easily solve a large range of software needs for the users.

We will first outline the reasons why WBAs reduce cost. Then we will describe three WBAs developed or being developed for the Thermosphere Ionosphere Mesosphere Energetics and Dynamics (TIMED) mission. The description will entail what the application is to do. The description will then give some details and comparisons between the development cycle of a WBA and a more traditional application.

2. Design Selection

WBAs can be used in a large range of applications since they are not dissimilar from the traditional applications that store and retrieve data for the user. In a more traditional approach, the data is stored in a large database. This database can either be written specific for the application or Commercial Off The Shelf (COTS). The user retrieves, edits, deletes, and adds to the database with an application. In many cases, the application has a graphical interface, especially if developed on PC's. Many older applications also rely on textual interfaces, which have forms to be filled out for data editing and manipulation. Reports are then generated to display the data. WBAs are just another approach in designing the interface for the user.

WBAs are not best for all software applications. Their strengths are in static data displays or interfaces to databases such as planning or assessment tools. They do not work well when there is rapidly changing information or rapidly changing graphics. However, this is possible using Java scripts and more powerful client machines. We will be addressing more of the static data types of applications.

The structure of a WBA is simple. First, there is a central server that houses the actual application and data. Next is the client that only has to supply a web browser. They can even exist on the same machine, but in most cases will reside on different machines.

3. Cost Benefits

There are two main reasons WBAs can have significant cost reducing benefits. The first is the cost of the operating system, development environment, web server, and database server can be nearly free or completely free. The software is referred to as Open Source Software/Free Software (OSS/FS). The second cost reducing benefit is a simpler development cycle, since the web browser provides the GUI or interface and the programming language is a script language tailored for data handling and web applications.

The first reason WBAs produce cost savings is from the environment of the application. This environment is the hardware, the operating system, the development tools, the web server, and the database server. For each of the software segments of the environment, OSS/FS is available and in wide use. Each software segment will be addressed and examples will be given for a suitable application. OSS/FS software is not the only choice available for the users and developers. Other types of software are COTS, or in−house written software. These will be also mentioned, but do not usually give a reduced cost for the application.

The OSS/FS operating systems are either a Linux distribution such as Red Hat, Debian, Slackware, or many others, or the modern BSD operating systems such as FreeBSD, NetBSD, or OpenBSD. At most, there is a medium and shipping cost that is usually below $100. These operating systems can also be easily downloaded from the provider's web site and burned onto CDs. The downside to these operating systems is little corporate assistance from the operating system's providers. Support can be supplied if needed and purchased. The lack of assistance is easily overcome with training and access to the web in the form of FAQ's and news chats. In essence the corporate assistance has moved away from the organization developing and selling the product to a large user base that will have intricate knowledge of the design and use of the operating system. In addition, these operating systems have a number of published books describing their use and administration. Furthermore, the operating systems are very close to or are Unix operating systems. Therefore, the in−house technical services will have knowledge and expertise in maintaining the operating systems. Lastly, these operating systems have been around for more than a decade. They have become simple to install, maintain, and operate as well as being robust operating systems. The user and developer will not have to invest much time or money in getting the operating system up and running.

Alternatively, COTS operating systems can be used.

These are the commercial Unix operating systems such as Solaris, Ultrix, or many others, or VMS from DEC, OS/X from Apple, or finally Windows 95/98/2000/XP from Microsoft. For a specific mission, there may not be costs associated with the operating system since the organization operating the satellite may have site license or large number of licenses for the operating systems. The cost may be accrued from maintenance cost or licensing fees.

Once the operating system is running and configured, the WBA can then have its environment built, such as the web server and the database server. The web server provides the connection between the user client and the WBA. The primary OSS/FS web server is Apache. Again, as in the operating systems, there is little corporate assistance on the web server, but licensing can be set up for support. If further help is still required, as in the operating systems, there is a large amount of help and expertise available on the web in the form of FAQs and news groups. Also, the web server has many books published on it describing its use and administration. This web server has been available for a long time and is reliable and easily maintained. Fortunately, the web server is simple to set up and maintain. The setup will not be complicated for most satellite operations centers, and this setup will not change significantly for the life of a mission. For most WBAs, a developer will not need to spend more than a few days setting up the web server and making it available to both the users and other developers. If a developer has prior experience on Apache, the set up could take as little as a few hours.

The COTS web servers are typically not suitable since they are expensive and designed for large simultaneous client connections. They can be used but are overkill.

After the operating system is running; the database server can be installed and setup, at the same time as the web server. Again, there are several OSS/FS database servers such as MySQL, PostgreSQL, or MSQL. Again, the free nature does not provide a large corporate assistance, but the use of FAQ's, simple training, and news groups allows for the administration of the database. In addition, these database servers have many books published describing their use and administration. Again, much like the operating systems, many of these databases have been around for close to a decade. In some of these cases, a license may be purchased from the developer if the database is used in a commercial or business environment. This cost is still low, less than $1,000. If there is a licensing fee, then technical help is available from the development team.

There are many COTS database servers, such as Access from Microsoft or Oracle may be available. These may have no direct cost to the individual missions as the organization may have licenses for the applications.

Lastly, the development environment can also be found with OSS/FS. This is not as much of a concern since either the operating system or the database server may provide most of the development environment needed for the WBA. There may be other tools such as configuration management or design cycle tools. The database server or the operating system, again, can also provide these. These tools may also be provided by the organization to unify the development process across all application developments.

The cost savings need not be limited to the software portions of the WBA. The hardware requirements typically will also be low. For this to be true, the application can not require large simultaneous client connections, which is typically true for operating satellites. The server will not need to have a high−end processor. This would come into play for a server that involves data analysis and storage for a department or company. For our approach, the number of client connections is small, simply the operations for a single, or cluster of satellites. The client connections base is quite often less than 10. A larger server would be required if multi−mission design is attempted. WBAs also can serve this roll, but become more troublesome if each mission has its own security guidelines. In addition, the server for the WBA has little display requirements; a very modest display can be used. This primarily needs to be used during installation and setup. From that time forward, all access to the server can be made remotely. This applies to the final application, development, and maintenance.

The second large cost reducing benefit from WBAs can be found in the development cycle. The savings are found for two primary reasons. The first reason is GUI development is not needed for WBAs. The web browser supplies the actual interface. The WBA only needs to write HTML to represent the interface. The second reason can be found in using a powerful high level scripting language to write the WBA in. The scripting language will have easy and rich libraries for text parsing, communications, and database manipulation that will reduce considerably the development time and can result in applications that can be easily maintained.

The web browser provides the user interface. Therefore, the WBA need only to generate HTML, which is standard ASCII text, to interface with the user. The web browser will correctly render an interface from the HTML. This interface can be simple or quite complex, but is still only driven from the HTML. The cost savings in this is the development of GUIs has now been reduced to understanding and generating HTML. The development of GUIs can be a significant portion of time for an application. In addition, since the web browser is providing the interface, the WBA has the appearance of a

cross−platform application. This means the client can be run on any machine that has a web browser suitable for the WBA. The client can have any of the operating systems listed above. It does not have the other components required by the server such as the web server, database server, nor the development environment. Since the client only needs to provide a web browser, its requirements are also modest.

The second reason for reduced development cost in WBAs is the use of powerful scripting languages for the WBA. These languages are typically Perl or PHP. Other languages can be used such as C, C++, Python, Ruby, or most other typical programming languages that can generate text output. Perl and PHP both provide very rich libraries for web development. In both cases, they are scripting languages, which are compiled at run time. The scripting languages allow for quick prototype development that can easily grow into the full application. It is easy to develop pieces of the application and bring the pieces together at the end. The individual pieces can be tested independently or at least have a simple wrapper. In addition the two languages have very good textual and data manipulation tools built in, as well as the tools to work with the database. This saves having to develop the tools.

Other additional cost saving benefits can also be found by using WBAs. These savings will not be as large as the two primary savings already described. Some savings can be found in the maintenance portion of the application. Here, the savings is typically found in adding functionality to the applications. The cause for savings here is for the same reasons as the development phase. The scripting languages easily allow new functions to be built in. The GUI is simple to change in allowing the new functions, in that the pages and forms can easily be changed to add the interface. The bug fixes or problem fixes part of maintenance will not see as much cost savings, but it may see some. This again comes from the use of the scripting languages. Since the languages have powerful text and data manipulation tools built in, the overall code should be smaller. Since, generally, the number of bugs is related to the size of the code, the number of bugs should be less. Furthermore, the scripting languages can be made to be easily human readable. As long as the original authors used good readable and maintainable practices in writing the WBA, they can be easily maintained. This is of course true with more standard development applications.

Another cost saving benefit from WBAs can be in training. This is not as significant as the two primary reasons, but may contribute some savings. The training is for both the developers and primarily the users. In both cases, the final user interface is in the form of a web page or form. The users of the WBA have by now become quite familiar on the operation and navigation of the web based pages and forms. The training is now focused on the specific tasks involved in operations. Little to no training on the actual user interface is required. This doesn't mean training for the application is no longer needed; what is no longer needed is how the user actually interfaces to the application, e.g., does he use menus, how are the menu items selected, etc. Removing the GUI development can reduce the training for the developers. This savings may be reduced by the developer's need to be trained in HTML, Perl, or PHP. There is no comparison in GUI development and scripting languages; GUI development requires more training and expertise.

4. Operations Benefits

In addition to cost savings listed above, WBAs also can provide other benefits. One simple benefit is to glue various aspects of operations into one more seamless package. Since the interface is a web browser, many different tools can now interact in new ways. For instance, during post−pass assessment of the telemetry, the operator discovers a range of telemetry that was not recovered. This range can now be fed into the planning application for new contact requirements. Once this is done, the range of data can be fed into the control application to build the necessary commands to recover the missing data. All this can now be done at the user's workstation.

Along with a more seamless application package, the WBA can be made for a distributed system. For this, the user need not always sit at only one console to perform his tasks. Any console with access to the server and which has an appropriate web browser can now be used. This can easily move the operations to the user's desk. However, this will add complications. The users will now have to be coordinated on what can and cannot be done at the desk. In addition, each person will have to know what other pieces of data other users are changing.

Security is a growing concern for everyone. Security can be easily built into a WBA with little additional cost. Furthermore, for most cases, security can be easily added to an existing WBA with little effort and cost. Security for a WBA can come in several forms. First is the security of the data. The database server can easily be set up that only an application local to the server machine can have access to the data. Since the WBA application is actually run on the server, this will lead directly to who can manipulate the data. Therefore most of the security is on who can have access to the server. The easiest security measure is a physically isolated network. This is common in secured military programs. A less isolated security measure is to use a firewall. Typically this would be at the gateway for an organization. This allows anyone in the organization access to the server. A further firewall could be put in place around the server. The clients could be

included within the firewall. In addition, the connections to the server can use Secure Socket Layer (SSL) and authentication. This means the data being transmitted is encrypted to prevent others from spying on the data. The web server can be set up to only allow connections from specific machines. Finally, passwords can be used to accept only allowable users to interact with the data. This can be expanded to allow only specific users to have full control of the data, while others can have simple query capabilities.

5. Examples

Parameter Input GUI

The first example offered is the Parameter Input GUI, or PIG. The PIG's sole purpose is to track the Guidance and Control's (G&C) parameter loads and states. It relies on parameter dumps as the input. A Perl script feeds the dumps into a MySQL database. The PIG's data is accessed from a web page where the user can select the processor and the parameter(s) to be viewed. The PIG tracks when the parameters were committed to non−volatile memory as well as resets when the non−volatile versions of the parameters are copied into working memory. Each parameter can be any dimension in size and are identified by an ID as well as a mnemonic style description. The PIG highlights changes in the parameters. Recent additions to the PIG generates a command script which can examine telemetry to verify the values of the parameter dumps are not different from the stored values. This is quite useful in comparing redundant sides as well as syncing engineering model test beds to the flight versions.

The PIG replaced a standard application, Parameter Archive Manager (PAM). PAM was written in Microsoft VBA. It read in parameter dumps and stored the values. The drawback to PAM was it displayed all the history for a given parameter. PAM quickly grew cumbersome shortly after the launch of TIMED when, due to anomalies on board, parameters were changed quickly. The operations team was offered two solutions, one was to make PAM more user friendly, two was to generate a WBA. The decision was to go with the WBA since both solutions would take similar time to develop and the WBA had the benefit of being able expand with later developments.

The PIG, once started took about 30 man−hours to finish. This time can be broken down by 10 man−hours for the data ingestion script for reading the parameter dumps into the database, then 20 man−hours for the web interface. PAM took about 180 man−hours to write, 60 man−hours for the input ingestion and 120 hours for the user interface.

Webular

The next example of a WBA for TIMED is Webular. Webular is in an early development phase. Currently, the TIMED operation center uses a Microsoft VBA application for scheduling contacts and events. TIMED is a routine science gathering spacecraft; it is not episodical, so most of the scheduling is done for contacts. The regular mission for TIMED will end in early 2004. A proposal to extend TIMED is in progress. For this proposal, the weekends and holidays will be handled entirely with lights out, or unstaffed contacts. The operations team currently uses lights out contacts for off shift contacts, but still staffs 7 days a week with 10 hour shifts. Recently, weather in Baltimore caused the operations team to rely on unstaffed contacts for several days. This caused problems since people had to plow their way into APL to plan the contacts and events. In addition, pointing data for the main dish is only available about 36 hours in advance.

The operation team realized a distributed scheduling system that can be accessed from home needs to be in place for the extended mission. Several options were available. First, using a PC anywhere type of tool that would allow the current PC with Schedular application to be remotely controlled. This would be quite difficult for two reasons. The first difficulty to overcome would be getting access through APL's and TIMED's firewalls. The maintainer of Schedular believes most remote control software relies on UDP to control a PC. This interface will be difficult to pass through the firewalls. The second difficulty in using remote control software would be bandwidth. Several members of the operations team only have dial−up access to the Internet. A PC controlling software would require much more bandwidth than available at 56k.

A WBA has been proposed to replace Schedular. The new software will be named Webular. It will have the same features as Schedular but with some look and feel differences caused by using a browser. One drawback, as pointed out by other operations team members, is the database tables will be more inaccessible. Currently, Schedular can be started in the design mode giving the users access to the tables. The data is easily visible much like a spreadsheet. There are tools to make the data also available, and we will look into these tools.

Webular is still being designed. The developers expect to need 8 man months to write Webular. Schedular took 10−12 staff months to write. The difference here is not as great but Webular will include additional features that were identified to assist entirely unstaffed weekend contacts.

Flexible Engineering Data System

The Flexible Engineering Data System (FEDS) was inspired by the speed and ease of the development of the PIG. The first step of the FEDS project was to modernize

the way spacecraft assessment is done today and in the future. The task of generating plots, display of archive telemetry, and generating assessment reports was in need of modernization. The FEDS project was developed to handle multiple spacecraft from multiple missions simultaneously. In our environment it is unheard of for separate missions to share hardware resources but with cost savings becoming more important, FEDS could make that change. For many spacecraft there are a few thousand telemetry points that comprise 90% of the data that is used in assessing the spacecraft health. It is this group of telemetry points that are the main focus of FEDS. A future step will be to add ties to real−time telemetry, which will allow the real−time assessment of real−time operations.

The heart of the FEDS project is a MySQL database that contains decommutated telemetry that can be raw values and/or values converted to engineering units. This database of values can be logically thought of as two databases, one for short−term values and a second for summarized values. The underlying structure is one database with many tables, but is simpler to envision as two databases. The short−term database contains every value available while the long−term database contains values summarized over a minute, hour, and day.

The short−term database is needed for close scrutiny of telemetry values and detailed reports. Some of the tables in this database can get very big, the best solution is to remove data as it ages, or have plenty of disk space available. The summarized version of the data will be available forever; this makes it easier to let go of the old data. Once old data is removed, it is possible to regenerate it from the raw telemetry if necessary. The generation of reports that require all telemetry values available is typically done in the near term so the loss of old data should have little impact on this. If the choice was made to never remove data from the short−term database this could be accommodated with a little work to keep the underlying database tables to a reasonable size. Currently we have a 15GB, and growing, table containing a portion of our telemetry data. This table has not caused any big performance problems for insertion of new and retrieval of old values and all on a 400MHz Intel processor. No old data is being removed from this table, it is being allowed to grow to see if MySQL or Linux will breakdown at some point.

The long−term database contains the summarized data over several time spans. The information in the summary would be the min, max, average, standard deviation as well as the number of values being summarized, and the values at the beginning and end of the time span. The summarized will best serve a look at data over a long term. A major use of this data would be in the generation of plots over a long period of time. When generating a plot whether on paper or in a browser there are only so many dots on the horizontal axis. If plotting a years worth of data for a single telemetry point there could be hundreds of thousands of values in the short−term database. With hundreds of values needing to be plotted when there is only one dot to represent them many values must be discarded or summarized. With a long−term database the summarized data is already available and ready to make the plot very quick to generate. It's only a matter of the plotting application choosing the summarized data that best fits in the requested time range.

Some may question putting telemetry values in a database but as part of normal operations the same telemetry is often being decommutated over and over for different plots and reports. It makes sense to decommutated the values once and have them on hand for quick use later. While FEDS does not do the decommutation of telemetry it can connect to the appropriate software to get decommutated values that are not in the database.

Some spacecraft contain tens of thousands of telemetry points and vast amounts of data for these spacecraft one would decommutate and save every value for every telemetry point. With the rapid strides in CPU power and disk space it is only a matter of time before this will become feasible; FEDS will be ready when this day comes.

FEDS contains generic tools for the display of telemetry value or plots on the fly but part of the beauty of using MySQL to hold the data, and PHP to generate web pages and plots, is that customized pages become very easy to develop. Different spacecraft have different requirements for reports and information dissemination; with the data in a database it becomes a simple task to generate web pages for reports covering varying time spans. The current methodology of generating daily, weekly, monthly plots and reports and printing them, now becomes a matter of developing web pages to do the same thing but with a more user friendly and flexible interface. A user can bring up a web browser at work, home or on his palm device at the beach and click through the standard set of pages of information or look at specific plots or data.

As alluded to earlier, the current software for generating reports does not lend itself to rapid development of reports. Currently reports are generated by C++ programs that pull raw telemetry data have it decommutated then massage the values and generate a paper report for a specific time period, which is printed. Using PHP to pull values from the database and massage the data into a report, which is output as an HTML page greatly reduces development time. The work to take the data and generate the information for the report is a similar between the two methodologies but PHP is a much easier language to work with. The web pages for a report are generated

dynamically with whatever time range the user desires. This gives the user a flexibility that can't be duplicated by the old system. With the reports almost instantaneously available via a web browser, the user not only get what he wants when he wants, he also get it where he wants.

FEDS is currently in its infancy, but it holds much promise for streamlining the way spacecraft assessment is done in the near future. With the database at the heart and PHP to generate web pages it is flexible and easy to customize to the different needs of different spacecraft. Currently FEDS is geared towards archived telemetry but there is a plan to allow it to receive real−time telemetry directly which it can then put into the database and generate strip charts of real−time data and supply to other things such as a real−time inference engine. Future upgrades of the FEDS system could easily bolt on other pieces to enhance the scope of its capabilities. One such product envisioned is an open source, real−time inference engine SCL, developed by Interface & Controls System of Columbia Maryland. With this product the world of real−time reaction to real−time events becomes possible. Once a powerful inference engine is incorporated into a system a new world of capabilities and possibilities can become a reality. The SCL system will add a Real−time Fault Detection Isolation and Resolution (FDIR) capability to the FEDS system. With an integrated script execution engine, SCL can detect data anomalies and trigger scripts to resolve the problem, and/or notify the spacecraft operations personnel and subsystem engineers in real−time via telephone, pager, or email (using user profiles to determine information delivery medium). These tools will make the FEDS system autonomous with operator control a web browser away.